Suffix automaton lecture notes

Brazilian Summer Camp 2018

Mikhail Tikhomirov

January 27, 2018

Suffix automaton

Suffix automaton (SA/DAWG) a directed acyclic graph with a dedicated *initial node* v_0, each arc is assigned with a single letter. For a SA of a string s the paths starting at v_0 are in one-to-one correspondence with substrings of s.

The *right context* $rc_s(v)$ of v with respect to s is a set of all strings t such that $v + t$ is a suffix of s. In the minimal SA of s each state corresponds to an equivalence class — *all* strings with the same value of $rc_s(v)$. If V is the largest string of a class, then all other strings represent several largest suffixes of V. A *suffix link* of a state points to another state corresponding to the largest suffix of V that lies in a different state. We define $suf(v_0) = -1$, where -1 is a virtual auxiliary state.

Things we store in a state st of SA:

- Transitions for all letters $st \to c$ (some of them may be undefined);
- The suffix link $suf(st)$;
- The length of the largest string in the state $len(st)$.

Algorithm

Let $v*$ be the state corresponding to the whole string s in an SA of s. We want to append letter c: $s \to s + c$. The algorithm:

1. Create the new vertex $v*'$ for the string $s + c$, put $len(v*') = len(v*) + 1$.
2. Let $v = v*$. While $v \neq -1$, and $v \to c$ is undefined:
   (a) $v \to c := v*'$;
   (b) $v := suf(v)$
3. If $v = -1$, set $suf(v*') := v_0$ and **finish**.
4. Now let $u = v \to c$. If $len(u) = len(v) + 1$, then $suf(v*') := u$, and **finish**.
5. Otherwise, create a new state u' — a copy of u. Set:
   (a) $len(u') := len(u) + 1$;
   (b) $suf(v*') := u'$;
   (c) $suf(u) := u'$.
6. While $v \to c = u$:
   (a) $v \to c := u'$;
   (b) $v := suf(v)$.
7. $v* := v*'$. **Finish**.

Note that the algorithm never creates more than two new states per phase, hence the number of states in the SA of s is at most $2|s|$. In fact, the total number of transitions in the SA of s is at most $3|s|$, and the complexity of this algorithm is $O(|s|)$.

Example applications

- To check if t is a substring of s, just follow the path corresponding to t and see if all transitions exist.
- To count the number of occurences of t, note that the answer is the number of paths leading to $v*$ from t. Since SA is an acyclic graph, we can compute the number of paths with DP in $O(|s|)$ time.
- The number of distinct substrings of s:
  1. The number of distinct substrings of s is equal to the number of distinct paths in the SA of s which can be found with DP.
  2. Another approach: note that each state v of the SA of s contains exactly $len(v) - len(suf(v))$ distinct strings, hence the sum of these values over all v is exactly the answer.